Discrete Optimization and Decision making project

Léos Coutrot - VR511700

August 15, 2024

Contents

1 Introduction

In modern urban logistics, efficient last-mile delivery systems are crucial for reducing costs, improving service levels, and minimizing environmental impact. This project explores a comprehensive delivery system design, tailored for a company operating within a city, which must serve a set of customers each requiring a package delivery. The system's design is complex, accounting for various constraints and objectives, including time windows for deliveries, capacity limitations, equity among delivery personnel, and environmental sustainability.

To achieve these goals, the project is divided into five main modules:

2 Module 1

The company operates a fleet of delivery vans that start from a common depot. The objective of this module is to minimize the total service time required to deliver packages to a set of customers $C = \{1, \ldots, \bar{c}\}$. The company has K vans with a maximum capacity W. The problem is represented on a complete directed graph $G = (V, A)$, where $V = C \cup \{0\}$ includes the depot and all customer nodes, and A represents the arcs connecting these nodes. Travel times between nodes $t_{ij}, \forall i, j \in A$ satisfy the triangle inequality.Each customer has a known package weight w_c and a service time s_c. The vans start from a common depot (node 0) and must return within a maximum allowed time $t_{\max}$. The delivery vans can only exit the depot once and each arc must be travelled at most once.

- C : Set of customers (indexed by c where $c \in \{1, \ldots, \bar{c}\}$).
- V : Set of nodes including the depot (indexed by i where $i \in C \cup \{0\}$).
- A : Set of arcs connecting nodes.
- K : Number of delivery vans.
- W : Maximum capacity of each van.
- t_{ij} : Travel time between nodes i and j.
- w_c : Package weight for customer c.
- s_c : Service time at customer c.
- $t_{\max}$: Maximum allowed time for the routes.

2.1 Decision Variables

- x_{ijk} : Binary variable that is 1 if van k travels from node i to node j, 0 otherwise.
- y_{ik} : Binary variable that is 1 if van k visits node i, 0 otherwise.
- t_{ik} : Time when van k arrives at node i.

2.2 Objective Function

Minimize the total service time which includes travel and service times:

$$\text{Minimize } Z = \sum_{i=0}^{\bar{c}} \sum_{j=0}^{\bar{c}} \sum_{k=1}^{K} t_{ij} x_{ijk} \tag{1}$$

$$+ \sum_{c=1}^{\bar{c}} \sum_{k=1}^{K} s_c y_{ck} \tag{2}$$

2.3 Constraints

2.3.1 Customer Visit Constraints

Each customer must be visited exactly once:

$$\sum_{k=1}^{K} y_{ck} = 1 \quad \forall c \in \{1, \ldots, \bar{c}\} \tag{3}$$

2.3.2 Van Capacity Constraints

The total weight of packages delivered by each van should not exceed its capacity:

$$\sum_{c=1}^{\bar{c}} w_c y_{ck} \leq W \quad \forall k \in \{1, \ldots, K\} \tag{4}$$

2.3.3 Flow Conservation Constraints

For each van, the flow conservation must hold:

$$\sum_{j=0}^{\bar{c}} x_{ijk} - \sum_{j=0}^{\bar{c}} x_{jik} = 0 \quad \forall i \in \{0, \ldots, \bar{c}\}, \forall k \in \{1, \ldots, K\} \tag{5}$$

2.3.4 Link Between x and y

Link x and y variables:

$$\sum_{j=0}^{\bar{c}} x_{ijk} = y_{ik} \quad \forall i \in \{1, \ldots, \bar{c}\}, \forall k \in \{1, \ldots, K\} \tag{6}$$

2.3.5 Subtour Elimination and Time Tracking

Ensure that the time constraints for visiting nodes and eliminating subtours are respected:

$$t_{0k} = 0 \quad \forall k \in \{1, \ldots, K\} \tag{7}$$

$$t_{jk} \geq t_{ik} + s_i + t_{ij} - M(1 - x_{ijk}) \quad \forall i, j \in \{1, \ldots, \bar{c}\}, \forall k \in \{1, \ldots, K\} \tag{8}$$

2.3.6 Time Constraints for Returning to Depot

Each van must return to the depot before the maximum allowed time:

$$t_{0k} + \sum_{i=1}^{\bar{c}} t_{i0} x_{i0k} \leq t_{\max} \quad \forall k \in \{1, \ldots, K\} \tag{9}$$

2.3.7 Depot Exit Constraints

Each van can only exit the depot once and each arc must be traveled at most once:

$$\sum_{j=0}^{\bar{c}} x_{0jk} = 1 \quad \forall k \in \{1, \ldots, K\} \tag{10}$$

$$\sum_{i=0}^{\bar{c}} x_{i0k} = 1 \quad \forall k \in \{1, \ldots, K\} \tag{11}$$

2.4 Python implementation

For this project, I solved the problem using the Pulp library, which is a library like Gurobi. I had to use this one instead of Gurobi because I had problems with my Gurobi license that I could not resolve. My code works pretty good, I managed to model this part without any problem. I took 3 vehicles and 10 customers for this part. The only issue is that the time taken to solve the problem is pretty long for rather small sets. It took 5 minutes to solve the problem. However, I still got this very satisfying result.

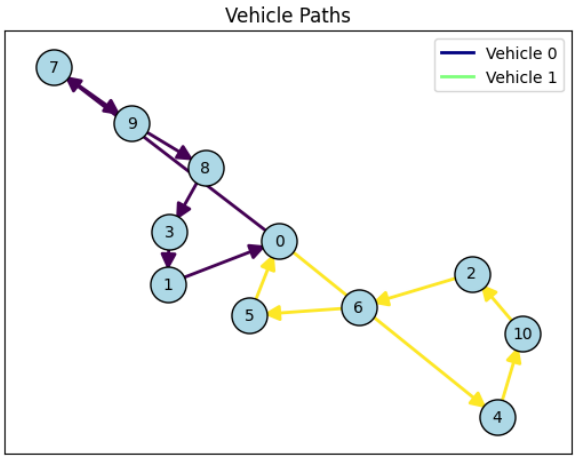

Figure 1: Result of my Python code for Module 1

We can notice that even though the company could have sent 3 vehicles here, it only sent 2 because it takes less overall time.

3 Module 2: Time windows and incorporating peculiar restrictions

This module introduces additional constraints based on peculiar restrictions and time windows. The company has a set of rules, represented as tuples $R = \{(i, j, \ell)\}$, where a tuple indicates that customer ℓ cannot be served by a vehicle that also serves both customers i and j. It also adds a time window for each costumers, the delivery must start within the time window $[a_c, b_c], \forall c \in C$

3.1 Updated Constraints

The main changes in the constraints for the updated model are detailed below. These changes account for time windows, forbidden routes, and time-based constraints:

3.1.1 Time Window Constraints

Each customer has an associated time window within which the service must be completed. The following constraints ensure that:

$$t_{j,k} \geq t_{i,k} + s_i + t_{ij} + a_j - M \cdot (1 - x_{ijk}) \quad \forall i, j \in \{1, \ldots, \bar{c}\}, \forall k \in \{1, \ldots, K\} \tag{12}$$

$$t_{i,k} \leq t_{j,k} - t_{ij} - s_i + M \cdot (1 - x_{ijk}) + b_j \quad \forall i, j \in \{1, \ldots, \bar{c}\}, \forall k \in \{1, \ldots, K\} \tag{13}$$

Here:

- $t_{i,k}$ is the time when van k arrives at customer i.
- s_i is the service time required at customer i.
- t_{ij} is the travel time from customer i to customer j.
- a_j and b_j are the earliest and latest allowable service times for customer j, respectively.
- M is a sufficiently large constant (Big M).

3.1.2 Forbidden Routes Constraints

Certain routes between customers are forbidden. The constraints ensure these routes are not used:

$$y[l,k] \leq 2 - y[i,k] - y[j,k] \quad \forall (i,j,l) \in R, \forall k \in \{1,\ldots,K\} \tag{14}$$

$$\tag{15}$$

Here:

- $y[i,k]$ is a binary variable indicating whether customer i is visited by van k.
- R is the set of forbidden routes, where (i,j,l) indicates that traveling from i to j makes traveling to l forbidden.

3.2 Python results

Here I just added to the code the changes. This time I took only 7 customers because it was way faster to compute (roughly 8 seconds). On each node we can see the time windows (depot has no time windows obiously):

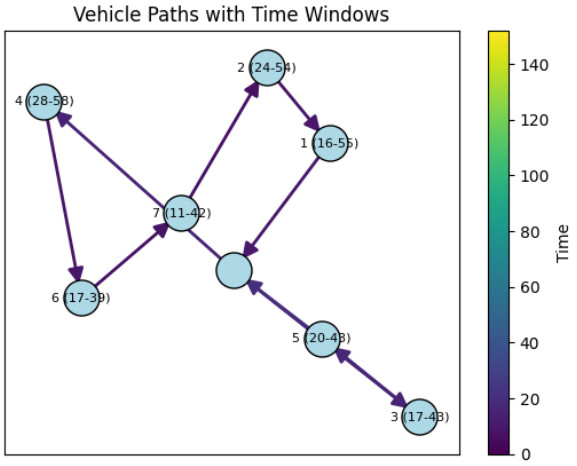

Figure 2: Result of my Python code for Module 2

Here we still observe that one car remains unused.

4 Module 3: Integration of Parcel Lockers

To enhance the flexibility and efficiency of the delivery system, this module introduces the use of parcel lockers across the city. Parcel lockers serve as intermediate drop-off points where packages can be left for customers to pick up, potentially reducing travel time and improving route efficiency. The problem is now defined over an augmented graph $G' = (V', A')$, where $V' = V \cup P$ and $P = \{1,\ldots,p\}$ is the set of parcel lockers.

To make this possible we don't need to change anything on our model constraints. We just have to do a filter : we consider each locker as a customer, if a customer is close enough to a locker, we always replace the customer by the locker and assign the sum of the weights packages to the locker. Once this is done, we can set their time window to $[0, t_{\max}]$, because a locker does not have a time window.

4.1 Python implementation

In this part I just added the code to compute the new set of customers. It allowed a a faster computing time and an overall better result with the objective function.

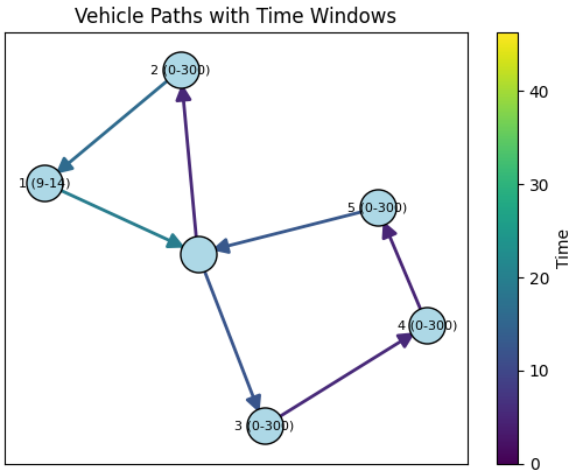

Figure 3: Result of my Python code for Module 3

5 Module 4: Equity in Workload Distribution

Recognizing the importance of equitable workload distribution among delivery drivers, this module introduces a secondary objective to minimize the mean absolute deviation (MAD) from the average workload. The workload is measured as the total time each driver spends completing their route and returning to the depot. This secondary objective is considered in a lexicographic manner, after the primary objective of minimizing the total service time.

5.1 Primary Objective Function

The primary objective remains to minimize the total service time, which includes the travel time and service time for each customer. This is formulated as:

$$\text{Minimize } Z_1 = \sum_{i=0}^{\bar{c}} \sum_{j=0}^{\bar{c}} \sum_{k=1}^{K} t_{ij} \cdot x_{ijk} + \sum_{i=1}^{\bar{c}} \sum_{k=1}^{K} s_i \cdot y_{ik} \tag{16}$$

where:

- x_{ijk} is a binary variable indicating if van k travels from customer i to customer j.
- y_{ik} is a binary variable indicating if customer i is visited by van k.
- t_{ij} is the travel time between customers i and j.
- s_i is the service time at customer i.

5.2 Secondary Objective Function: Workload Balancing

The secondary objective aims to minimize the mean absolute deviation (MAD) of the return times from the depot for each driver. The MAD is defined as:

$$\text{Minimize } Z_2 = \frac{1}{K} \sum_{k=1}^{K} \left| T_k - \bar{T} \right| \tag{17}$$

where:

- T_k is the total return time for van k.
- $\bar{T}$ is the average return time across all vans, calculated as:

$$\bar{T} = \frac{1}{K} \sum_{k=1}^{K} T_k \tag{18}$$

5.3 Lexicographic Order of Objectives

The optimization process uses a lexicographic approach:

- The primary objective Z_1 is given the highest priority and is optimized first.
- The secondary objective Z_2 is optimized only if the primary objective has been solved within 5% of its optimal value.

The 5% degradation threshold is incorporated as:

$$Z_1^{\text{new}} \leq 1.05 \cdot Z_1^{\text{optimal}} \tag{19}$$

where:

- Z_1^{new} is the value of the primary objective function in the solution considering both objectives.
- Z_1^{optimal} is the optimal value of the primary objective function when considered alone.

5.4 Python implementation

Just like for modules 1 and 2, I gave Pulp the new constraints and added the second objective function.

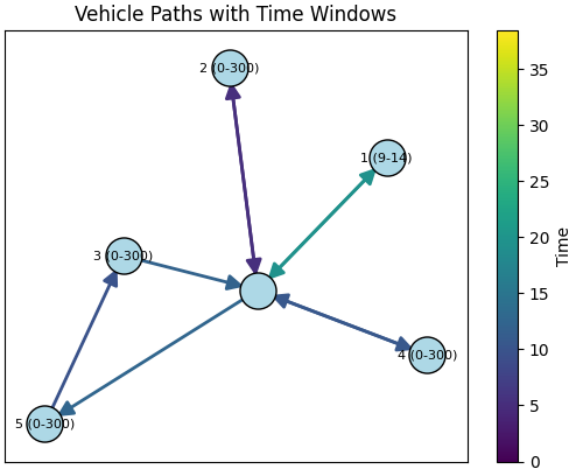

Figure 4: Result of my Phyton code for module 4

We can see that now all the vehicles are used by the company.

6 Conclusion

I did not manage to do module 5. However, I am really happy of what I did for the 4 other modules. I think the computation time could be an issue if I had to try my code on a real life datset with thousands of customers. It might be because I use Pulp library, which is slower than Gurobi. However, I think that even Gurobi would be slow, so maybe an approach with dynamic programming would be better. For example, an approach like the MACS-DVRPTW (Multiple Ant Colony System for Dynamic Vehicle Routing Problem with Time Window) would be more efficient I believe. But this is in my opinion a hard task that would deserve a whole other project to implement and detail correctly.

Even though my code is not the fastest, I think it could still be usable for a lot of cases. Not all companies have to compute how to deliver thousands of packages every day, a lot of smaller companies could use it. Also, we have to keep in mind that the company only has to perform the computation once everyday. So if execute the code everyday at midnight, we would have more than enough time to complete the computation.

References